Project Review and Completion for Django Backend

Recap of Previous Steps: Previously set up a Django project with the following structure:

django-admin startproject sensor_project

sensor_project

- manage.py
- settings.py
- urls.py
- asgi.py
- wsgi.py

Current Focus: Interactions Between Modules

Step 1: Connecting Django with MySQL

Open settings.py and configure the database settings to connect with your MySQL database. Modify the **'DATABASES'** section to include your database details and ensure your MySQL password is correctly input.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp',
    'rest_framework',
]
```

This list contains the installed applications in a Django project, each of which is a Python package usually located within the project directory. Here are the purposes of each application:

- django.contrib.admin: Provides the Django admin interface for site management.
- django.contrib.auth: Offers an authentication system, including user login, registration, and permissions.
- django.contrib.contenttypes: Allows the creation of generic relationships, useful in the permissions system.
- django.contrib.sessions: Manages user sessions, supporting the storage of user data across multiple requests.
- django.contrib.messages: Provides a messaging framework that allows storing messages across requests.
- django.contrib.staticfiles: Assists in managing static files (such as CSS, JavaScript, images).
- **myapp: A custom application created to contain the project's business logic.**
- **rest_framework: Django REST framework, a powerful and flexible toolkit for building Web APIs.**

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'sensor_data',
        'USER': 'root',
        'PASSWORD': '123Wjb456+',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

This dictionary defines the database configuration used by the Django project. Here, a MySQL database is configured:

- ENGINE: **The database engine, django.db.backends.mysql, indicates the use of MySQL**.
- NAME: **The name of the database, which is sensor_data in this case**.
- USER: **The username for connecting to the database, which is root here**.
- PASSWORD: **The password for connecting to the database, which is 123Wjb456+**
- HOST: The address of the database server; localhost indicates a local server.
- PORT: The port number for the database server, with MySQL defaulting to port 3306.

Step 2: Configuring apps.py in the Sensor App

Navigate to the apps.py file within the sensor folder, **named 'myapp'**. No changes are needed for the folder name.

```
myapp > apps.py > ...
1   from django.apps import AppConfig
2
3
4   class MyappConfig(AppConfig):
5       default_auto_field = 'django.db.models.BigAutoField'
6       name = 'myapp'
7
```

This file defines some configurations for the Django application. Specifically, it defines a configuration class MyappConfig, which inherits from django.apps.AppConfig. The primary purpose of this apps.py file is to set up basic information for the myapp application, including the application name and the default auto field type. These configurations help Django recognize and correctly handle various models and functionalities within the application.

default_auto_field = 'django.db.models.BigAutoField'

This line of code specifies the default auto field type for models in the application. BigAutoField is a large integer auto-incrementing field type, typically used for primary keys. This means that if a model within the application does not explicitly specify a primary key type, BigAutoField will be used by default.

Step 3: Register Models in admin.py

Open admin.py and register your models. Ensure the models.py file contains a class named **'Sensor'**. If it does, **register the 'Sensor' model in admin.py, ensuring the name matches**. Do the same for the **'Measurement' model**.

```
myapp > admin.py
1   from django.contrib import admin
2   from .models import Sensor
3
4   # Register your models here.
5   admin.site.register(Sensor)
```

This file serves to register the Sensor model with the Django admin site, enabling its management through the admin interface. By registering the Sensor model, you can utilize Django's built-in admin functionalities to manage the application's data.

admin.site.register(Sensor)

This line of code registers the Sensor model with the Django admin site. The **admin.site.register() function takes a model as an argument and adds it to the admin interface.** This allows you to view, add, modify, and delete Sensor model entries from the admin interface.

Step 4: Checking serializer.py

In the serializer.py file, ensure all fields correspond to your Excel file, with correct case sensitivity.

The serializers.py file defines two serializer classes, SensorSerializer and MeasurementSerializer, which are used to convert instances of the Sensor and Measurement models into a serializable format such as JSON. These serializers are very useful when building REST APIs, as they ensure the accuracy and consistency of data during transmission.

The Meta class specifies the fields that need to be included for the Sensor and Measurement models.

Step 5: Configuring urls.py in myapp folder

Open 'myapp/urls.py' and adjust the URL paths. Register the 'Sensor' and 'Measurement' models with the router. **Ensure the names match the models exactly**, maintaining the correct case.

myapp > urls.py > ...

```python
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import SensorViewSet, MeasurementViewSet
from .views import index

router = DefaultRouter()
router.register(r'Sensor', SensorViewSet)
router.register(r'Measurement', MeasurementViewSet)

urlpatterns = [
    path('api/', include(router.urls)),
    path('', index, name='index'),
]

```

The urls.py file defines the project's URL routing configuration by utilizing Django and Django REST framework's routing mechanisms. It includes two main components:

- **Automatically generated RESTful API endpoints**: These endpoints handle the CRUD operations (Create, Read, Update, Delete) for sensor and measurement data.
- **A simple homepage view**: This view provides a response to the root URL.

These configurations enable the project to offer both API interfaces and handle regular web page requests.

Step 6: Adjusting views.py

Modify views.py to include two viewsets: 'SensorViewSet' and 'MeasurementViewSet'. Both should get all objects.

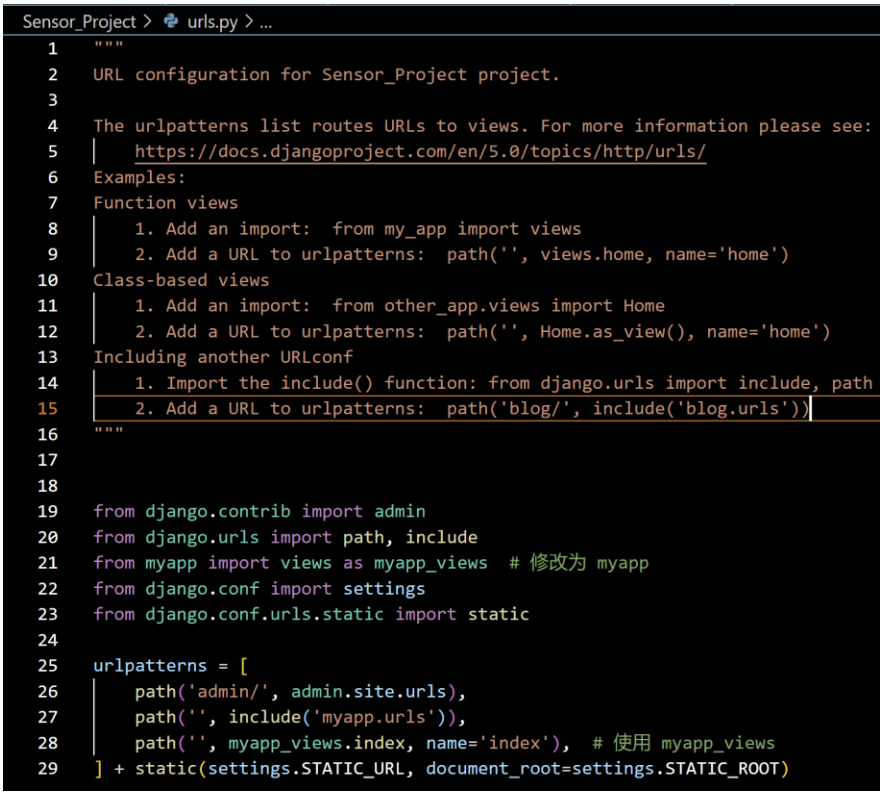
myapp > views.py > index

```python
from rest_framework import viewsets
from .models import Sensor, Measurement
from .serializers import SensorSerializer, MeasurementSerializer
from django.shortcuts import render

class SensorViewSet(viewsets.ModelViewSet):
    queryset = Sensor.objects.all()
    serializer_class = SensorSerializer

class MeasurementViewSet(viewsets.ModelViewSet):
    queryset = Measurement.objects.all()
    serializer_class = MeasurementSerializer


def index(request):
    return render(request, 'frontend/public/index.html')
```

The views.py file defines two view sets, **SensorViewSet and MeasurementViewSet**, **which handle API requests for sensor and measurement data, respectively. Each view set inherits from ModelViewSet, providing standard CRUD operations.** The file also defines a simple index view function that handles requests to the root URL (/) and returns the index.html template located in the frontend/public/ directory.

Step 7: Finalizing urls.py in the Sensor_Project Folder

Open 'Sensor_Project/urls.py' and adjust it accordingly. Running the server **python manage.py runserver**



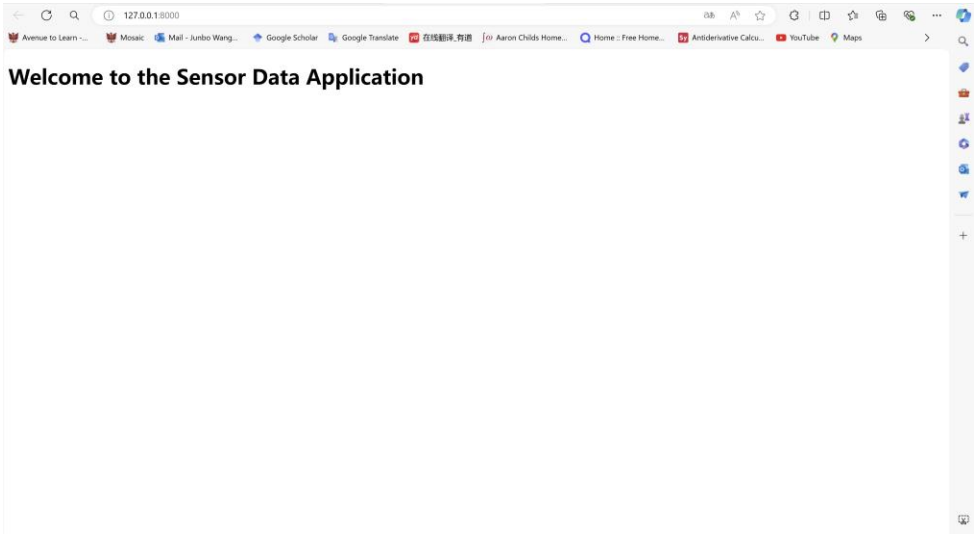
```
Sensor_Project > urls.py > ...
"""
URL configuration for Sensor_Project project.

The urlpatterns list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/5.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""


from django.contrib import admin
from django.urls import path, include
from myapp import views as myapp_views  # 修改为 myapp
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls')),
    path('', myapp_views.index, name='index'),  # 使用 myapp_views
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

urls.py file configures the project's URL routing by defining the urlpatterns list. It includes:

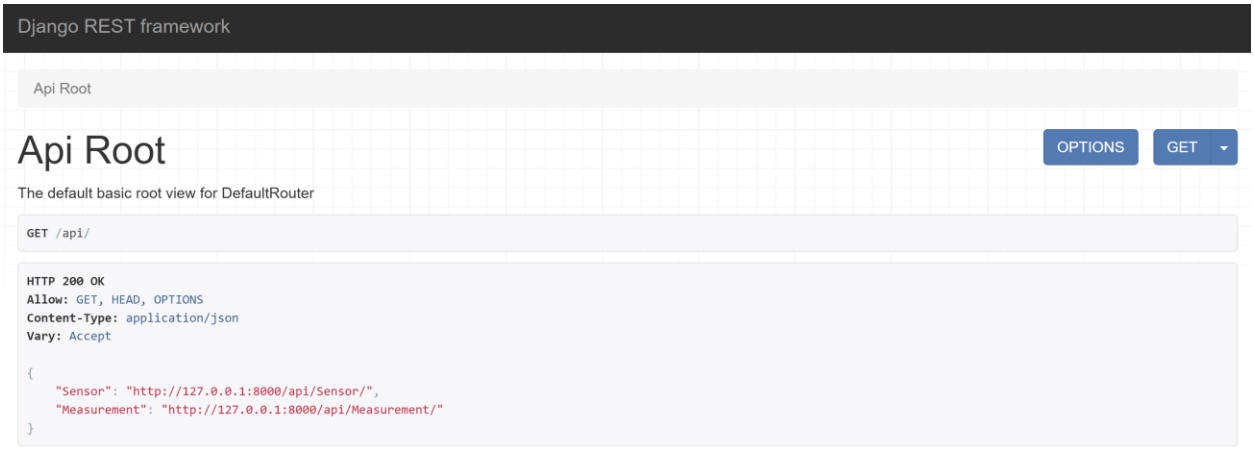
- path('admin/', admin.site.urls), : **The URL routing for the admin interface**.
- path('', include('myapp.urls')), : **Maps the root URL ('') to the URL configuration of the myapp application**.
- path('', myapp_views.index, name='index'), : **Maps the root URL to a homepage view**.
- + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT) : **Configures the URL routing for static files to serve them in the development environment**.

These configurations ensure that the Django project can correctly route to the appropriate views and handle static files.

http://127.0.0.1:8000/



Test 127.0.0.1:8000/api



Django REST framework

Api Root / Sensor List

Sensor List

OPTIONS GET

```
GET /api/Sensor/
```

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "Sensor_ID": "DHT11",
        "Location": "bedroom",
        "Sensor_Key": 1
    },
    {
        "Sensor_ID": "DHT22",
        "Location": "balcony",
        "Sensor_Key": 2
    },
    {
        "Sensor_ID": "DHT11",
        "Location": "First Floor",
        "Sensor_Key": 3
    },
    {
        "Sensor_ID": "DHT22",
        "Location": "bathroom",
        "Sensor_Key": 4
    },
    {
        "Sensor_ID": "DHT11",
        "Location": "backyard",
        "Sensor_Key": 5
    }
]
```

Django REST framework

Api Root / Measurement List

Measurement List

OPTIONS GET

```
GET /api/Measurement/
```


```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "ID": 1,
        "Temperature": 28.5,
        "Humidity": 56.0,
        "Timestamp": "2024-06-21T11:24:32Z",
        "TimeZone": "morning",
        "Temperature_Change_Rate": "no change",
        "Humidity_Change_Rate": "no change",
        "Comfort_Index": -2.0541,
        "Location": "bedroom",
        "Sensor_ID": 1,
        "Comfort_Level": "High"
    },
    {
        "ID": 2,
        "Temperature": 28.5,
        "Humidity": 56.0,
        "Timestamp": "2024-06-21T11:24:34Z",
        "TimeZone": "morning",
        "Temperature_Change_Rate": "no change",
        "Humidity_Change_Rate": "no change",
        "Comfort_Index": -2.0541,
        "Location": "bedroom",
        "Sensor_ID": 1,
        "Comfort_Level": "High"
    },
    {
```